**12 –Merge Conflicts**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

**Name:**

Top of FormThis topic focused on merge conflicts, why they arise and how they can be resolved. We saw that maintainers are regularly merging changes from contributor’s pull requests into the upstream main branch. If those changes are merged after you created your feature branch, it is possible for a merge conflict to arise. If your changes and those merged into main affect different parts of the project, then the maintainers will still be able to merge your changes automatically. However, if the changes merged into main and the changes in your feature branch alter the same code or documentation, then a merge conflict will occur. Usually, you as the contributor and the expert on your work will be asked to resolve the merge conflict so that the maintainers can merge your pull request automatically. We saw that this can be done using a graphical merge tool (e.g. meld). In completing the most recent homework, you will have created a pull request that, after other work was merged into the upstream main branch, has resulted in a merge conflict. This set of activities takes you through the process of resolving that merge conflict.

**Merge conflict concepts:**

1. Consider the 3-way merge shown below. As shown in the right-hand pane, the maintainers have merged commits into the main branch that fix several bugs that existed in the program. The left-hand pane shows a contributors feature branch with has been changed to use more descriptive variable names, but has not fixed the bugs. The center pane shows the best common ancestor of the feature and main branches.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | total=0  count=0  read n  while count > n:  read m  total=total+m  count--  average=count/total |  | tot=0  count=0  read n  while count > n:  read m  tot=tot+m  count--  ave=count/tot |  | tot=0  count=0  read n  while count < n:  read m  tot=tot+m  count++  ave=tot/count |  |
|  | **Feature Branch** |  | **Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

a. Describe in a sentence the computation that the program above is trying to perform. Note: The program in the main branch functions correctly, the others contain bugs.

b. Use the highlight tool to mark the lines in the feature branch and/or the main branch above as indicated below. Use the example in the slides as a guide for the highlighting.

i. Highlight all lines containing non-conflicting changes in blue.

ii. Highlight all lines that contain conflicting changes in red.

Nothing is required here. Just be sure to highlight the code above as indicated.

c. Would the feature branch above be able to be merged automatically by the project maintainers? Briefly explain your answer.

d. Give a merged version of the above program that combines the feature branch and the main branch such that the result both performs the desired computation and uses the more descriptive variable names.

2. Now consider the following 3-way merge:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | r = 15  pi = 3.1415927  rsq = r\*\*2  a = pi\*rsq  print a |  | r = 15  pi = 3.14  rsq = r\*2  a = pi\*rsq  print a |  | r = 15  pi = 3.14  rsq = r\*2  area = pi\*(rsq/2)\*\*2  print area |  |
|  | **Feature Branch** |  | **Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

a. Describe in a sentence the computation that the program above is trying to perform. Note: The Common Ancestor contains a bug that is fixed in different ways by the main branch and the feature branch.

b. Highlight the above merge in the same way you did in question #1b.

Nothing is required here. Just be sure to highlight the code above as indicated.

c. Briefly explain why this merge could be performed automatically.

d. Fill in the Result column to show the result of an automatic merge of the feature and main branches.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | r = 15  pi = 3.1415927  rsq = r\*\*2  a = pi\*rsq  print a |  |  |  | r = 15  pi = 3.14  rsq = r\*2  area = pi\*(rsq/2)\*\*2  print area |  |
|  | **Feature Branch** |  | **Result** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

e. Look carefully at your result in part d. Will the resulting program perform the computation correctly? Briefly explain why or why not.

f. What does your answer to part e suggest to you about automatic merge operations?

**Getting the KitClient Started Again:**

Like the last several activities, you will be working within the *KitClient* for this activity.

3. Revisit one of the prior activity sheets to find the docker run command that you used to start the KitClient. Start the KitClient and connect to it with Tiger VNC or noVNC.

Nothing is required here, but you must have the KitClient running to work with your local copy of the FarmData2 repository.

**Synch with the Upstream main:**

In class we saw that the maintainers had merged some changes into the upstream main after you had created your feature branch. The changes that were merged were specifically designed to conflict with the changes required for each of the Round2 issues. Thus, the pull request that you made at the end of the previous activity will now contain conflicts that prevent it from being merged automatically. This situation is shown in Figure 1, where the maintainers have merged the fuchsia commit into the main branch.

Figure 1 - Conflicting Upstream Changes

The activities in the remainder of this section will have you confirm that you are out of synch with the upstream and that your pull request cannot be merged automatically. It will then have you synch the main branch in your local and origin repos with the upstream, so that you can resolve the merge conflict.

4. Which of the commits in Figure 1 (i.e. which colors) could contain merge conflicts after the fuchsia commit was merged?

5. Visit the main page for your origin repo on GitHub and make sure that the main branch is active.

How can you tell from this page that there have been changes to the upstream main branch that you have not yet synched?

6. Now find your pull request in the upstream repo on GitHub. You should see that your pull request cannot be merged automatically.

a. How does GitHub indicate that your pull request cannot be merged automatically?

If GitHub indicates that your pull request can be merged automatically follow the instructions in Appendix A at the end of this activity sheet and then return to this question when you have a pull request that cannot be merged automatically.

b. Briefly explain in a sentence or two of your own words, why your pull request cannot be merged automatically. Hint: Think Figure 1!

7. Use what you learned in the prior activity to synch the main branch of your local and origin repos with the upstream. Don’t forget to switch to your main branch first!

a. Show the commands that you used and their output as your answer for this question.

b. Check that you are now synchronized by checking that the main branch on your origin repo on GitHub “is up to date…” (See #5). Then also use git status to check that the main branch in your local repo is up to date with your origin. If the main branch in your local or origin repos are not up to date, double check your synchronization commands in part a and try again.

Nothing is required here, but the main branches in your local and origin repos must be up to date before continuing.

**Merge main Branch into Feature Branch:**



Figure 2 - Merge main into feature branch

Your local main branch now contains the changes that were made to the upstream main branch. The next step is to merge the changes from the main branch into your feature branch. This is similar to the example from class, which is illustrated in Figure 2.

8. Before you can merge main into your feature branch, your feature branch must be the active branch.

Change to your feature branch and confirm that it is now the active branch. Give the commands that you used and their output here.

9. The git merge <branch name> command will attempt to merge the specified branch into the active branch.

a. Give a command that will merge the main branch into your feature branch.

b. Give the full output that your command from part a generated.

10. The output from #9.b should indicate that the automatic merge failed due to a conflict. If your merge command did not fail due to a conflict, return to question #6a and follow the instructions from there again.

a. In which file is the conflict located?

b. Use the cat command to display the file containing the conflict.

Find the part of the file that contains the *raw conflict information* and copy and paste it here. Be sure to include the chevrons at the top and bottom of the conflict information.

11. Answer the following questions by highlighting the output that you gave in question #10.b.

a. Highlight the content from before any changes were made in green.

b. Highlight the content showing the changes that the maintainers merged into the main branch after you made your feature branch in yellow.

c. Highlight the content showing your changes in blue.

Nothing is required here, just ensure that you have properly highlighted the output that you gave in question #10b.

12. Sometimes when you try to merge, and it fails due to a conflict you may want to undo the attempted merge. The git merge --abort command will undo a failed merge.

Undo the merge that you started in question #9 and use the cat command to display the file that contains the conflict again.

Briefly explain how you can tell that the failed merge has been undone?

**Resolving a Merge Conflict:**



Figure 3 - Resolving a Merge Conflict

This section will walk you through the process of resolving the merge conflict using the graphical merge tool *Meld*. The flow of this process is illustrated in Figure 3. You will switch to your feature branch, perform the merge, use Meld to resolve the resulting conflict, and then stage and commit the merged content.

13. Ensure that you are on your feature branch. Then issue the command to merge the main branch into your feature branch (see #9). The merge should once again fail because of the conflict. Use the cat command to confirm that the raw merge conflict information now appears in the conflicted file again.

No answer is necessary here, but you need to ensure that the conflicted file contains the raw merge conflict information before continuing.

14. When a merge fails, git places the raw merge conflict information into the conflicted files as we have seen. The git mergetool command will launch a merge tool (e.g. Meld) so that the conflict can be resolved. The KitClient you are using has been configured so that the

git mergetool command launches Meld.

Issue the git mergetool command.

Paste a screen capture of the Meld window here before doing anything to its content.

15. When using Meld you resolve conflicts by modifying the center pane so that it appears as desired. You can click the arrows to move content between the panes and you may also directly edit the center pane.

Use Meld to resolve the conflict so that the merged result will contain:

* your changes where there is a conflict (red highlights).
* all of the other non-conflicting changes from the main branch (blue highlights).

Click “Save” in Meld to save your changes.

Paste a screen shot of the Meld window showing the merged result here.

16. Close the Meld tool, issue a git status command and give the output here.

17. From #16 you can see that you now have uncommitted changes. Those changes are all of the changes you made to the common ancestor when performing the merge in Meld.

Stage and commit the changes you made in resolving the conflict. Be sure to use a meaningful commit message.

Give the commands that you used and their output here.

**Update your Pull Request:**

At this point you have resolved the merge conflict on the feature branch in your local repository. What is left is to push that branch to your origin. When you do so, GitHub will automatically update your pull request to the upstream for that branch.

18. Push your modified feature branch to your origin. Give the command you used and its output here.

19. Now visit the upstream repo and find your pull request. It should now be able to be merged automatically. If your PR cannot be merged automatically you have not correctly resolved the merge conflict, return to question #8 and try again.

**Reflection and Summary:**

20. The Git and GitHub activities you have completed and the KitClient are part of a collaborative project to produce materials for teaching git and GitHub at other colleges and universities. To help with that effort we are collecting data about the experiences students are having using the materials and the KitClient.

Please complete the following very short (< 5 minute) survey hosted at Drexel University to help us improve these materials and the KitClient for other students:

<https://drexel.qualtrics.com/jfe/form/SV_0PY4paHUi2CavH0>

21. Complete the table below by filling in the right-hand column with the commands that accomplish the task listed on the left. Use the <…> notation appropriately to indicate parameters that need to customized for each use. Note that the tasks listed are in approximately the same order as they appear in this activity.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task to Complete** | **Git Commands** |  |
|  | Merge changes from one branch into another branch |  |  |
|  | Undo a merge that failed due to conflicts |  |  |
|  | Launch the configured graphical merge tool |  |  |
|  |  |  |  |

**Optional Additional Practice with Merge Conflicts:**

22. This section is optional but will provide you with more practice with Git, GitHub and with resolving merge conflicts if you would like it.

a. Use the following command to get a feature branch from the upstream into your local repo:

git fetch upstream merge-conflict-practice

The merge-conflict-practice branch was created so that it contains some conflicts with the upstream main branch. When you have fetched this branch, think of it as if you had created it from main, then made some changes that resulted in conflicts with the upstream main branch. This leaves you at the point where you are trying to merge main into this feature branch and resolve the conflicts.

b. Make the merge-conflict-practice branch active and try to merge main into it. Give the commands that you used and their output here.

c. The merge-conflict-practice branch contains a few conflicts so the merge you attempted in part b should fail. Resolve the conflicts using the changes in the merge-conflict-practice branch (red highlights) and also accept all of the non-conflicting changes from the main branch (blue highlights).

Nothing is required here, but you must resolve the conflicts before continuing.

d. Stage and commit the changes you made to resolve the conflicts to you feature branch. Be sure to use a meaningful commit message. Give the commands that you used and their output here.

d. Push the merge-conflict-practice branch to your origin and create a pull request to the upstream. What is the title and number of your pull request?

**Appendix A:**

Skip this section if you were not directed here from question #6.

If you were directed here from question #6, then the pull request you created for your Round2 issue is showing that it can be merged automatically. The most common reason that this happens is that you synchronized your local and origin main branches after the instructor merged the conflicting changes into the upstream main branch. This could have happened if you did not complete the previous assignment on time. The instructions in this section will help you to resolve that issue, but will require a little additional work on your part.

A.1. Ensure that you are in your local repository in a Terminal. Use the following commands to restore your main branch to the state before the conflicts were introduced:

git switch main

git reset --hard 85b82775fa48c1b40727c9782dcd2f4a937ac6d1

You should see the output:

HEAD is now at 85b8277 Do not display directory during execution.

A.2. You will now need to:

* Repeat the steps you used to fix your Round2 issue:
  + make a new feature branch
  + switch to your feature branch
  + make the changes necessary to address your Round2 issue
  + stage and commit your changes to your feature branch
  + Push your feature branch to your origin
  + Make a pull request to the upstream for your new feature branch
    - When you look at this pull request it should indicate that it cannot be merged automatically.

A.3. Return to question #6 and continue with this assignment.

**Optional:** To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.

**Acknowledgements:**

Some materials, questions and resources have been adapted from activities posted on foss2serve.org:

* <http://foss2serve.org/index.php/Git:_GitHub_Issues_and_Pull_Requests>
* <http://foss2serve.org/index.php/Git:_GitHub_Workflow_Activity>
* <http://foss2serve.org/index.php/Work_Locally_with_Git_from_the_Command_Line_(Activity)>